IEICE TRANS. COMMUN., VOL.E92–B, NO.6 JUNE 2009

PAPER

Fast Packet Classification Using Multi-Dimensional Encoding

Chi Jia HUANG[†], *Nonmember and* **Chien CHEN**[†a)], *Member*

SUMMARY Internet routers need to classify incoming packets quickly into flows in order to support features such as Internet security, virtual private networks and Quality of Service (QoS). Packet classification uses information contained in the packet header, and a predefined rule table in the routers. Packet classification of multiple fields is generally a difficult problem. Hence, researchers have proposed various algorithms. This study proposes a multi-dimensional encoding method in which parameters such as the source IP address, destination IP address, source port, destination port and protocol type are placed in a multi-dimensional space. Similar to the previously best known algorithm, i.e., bitmap intersection, multi-dimensional encoding is based on the multi-dimensional range lookup approach, in which rules are divided into several multi-dimensional collision-free rule sets. These sets are then used to form the new coding vector to replace the bit vector of the bitmap intersection algorithm. The average memory storage of this encoding is $\Theta(L \cdot N \cdot \log N)$ for each dimension, where L denotes the number of collision-free rule sets, and N represents the number of rules. The multi-dimensional encoding practically requires much less memory than bitmap intersection algorithm. Additionally, the computation needed for this encoding is as simple as bitmap intersection algorithm. The low memory requirement of the proposed scheme means that it not only decreases the cost of packet classification engine, but also increases the classification performance, since memory represents the performance bottleneck in the packet classification engine implementation using a network processor.

key words: *router, packet classification, multi-dimensional encoding, bitmap intersection, network processor*

1. Introduction

The accelerated growth of Internet applications has increased the importance of the development of new network services, such as security, virtual private networks (VPN) and quality of service (QoS). These mechanisms generally require the router to categorize packets into different classes, called flows. The categorization function is packet classification.

An Internet router categorizes incoming packets into flows, utilizing information contained in the packet header and a predefined rule table in the router. A rule table maintains a set of specific rules based on the packet header fields, such as the network source address, network destination address, source port, destination port and protocol type. The rule field can be a prefix (e.g. a network source/destination address), a range (e.g. a source/destination port) or an exact number (e.g. a protocol type).

When a packet arrives, the packet header is extracted first and then compared with the corresponding fields of rules in the rule table. A rule matching all corresponding fields is considered as a matched rule. The packet header is compared with every rule in the rule table, and the matched rule with the highest priority yields the best-matching rule. Finally, the router performs the action associated with the best-matching rule.

The d-dimensional packet classification problem is formally defined as follows. The rule table has a set of rules $R = \{R_1, R_2 \ldots, R_n\}$ over d dimensions. Each rule comprises d fields $R_i = \{F_{1,i}, F_{2,i} \ldots, F_{d,i}\}$, where $F_{j,i}$ denotes the value of field j in rule i. For example, the field could be the source IP address, destination IP address, port number or protocol type. Each rule also has a priority. A packet P with header field $(p_1, p_2, \ldots, p_d)$ matches rule R_i if all the header fields p_m, where $1 \leq m \leq d$, of the packet match the corresponding fields $F_{m,i}$ in R_i. IF the packet P matches multiple rules, then the highest priority rule is returned.

The general packet classification problem can be viewed as a point location problem in multidimensional space [1]. Rules have a natural geometric interpretation in d dimensions. Each rule R_i can be considered as a "hyper-rectangle" in d dimensions, obtained from the cross-product of $F_{j,i}$ along each field. The set of rules R can thus be considered as a set of hyper-rectangles, and a packet header denotes a point in d dimensions. The bitmap intersection algorithm [2], proposed by Lakshman and Stiliadis, converts the packet classification problem into a multidimensional range lookup problem. Bitmap intersection constructs bit vectors for each dimension. A simple "multidimensional search," consisting of a logical AND operation, is then applied to multiple bit vectors. A good packet classification algorithm must categorize packets quickly with minimal memory storage requirements. Since the memory storage taken by the bitmap intersection algorithm quadruples every time the number of rules doubles, the bitmap intersection algorithm is not appropriate to apply to large rule numbers.

P^2C [12], instead utilizes the concept of "independent field search," in which a one-dimensional encoding is applied to each independent field to decrease the storage requirement. The P^2C encoding is based on the primitive ranges, which are intervals of field values. Figure 1 illustrates an example of P^2C one-dimensional encoding style. However, to implement multi-dimensional search efficiently, the P^2C approach must search independent fields in parallel. Hence, a TCAM is proposed to speed up the search. The TCAM entries include new coding vectors, which are

Manuscript received July 11, 2008.
Manuscript revised December 21, 2008.

[†]The authors are with the Department of Computer Science, National Chiao Tung University, HSINCHU 30050, Taiwan, ROC.
a) E-mail: chienchen@cs.nctu.edu.tw

DOI: 10.1587/transcom.E92.B.2044

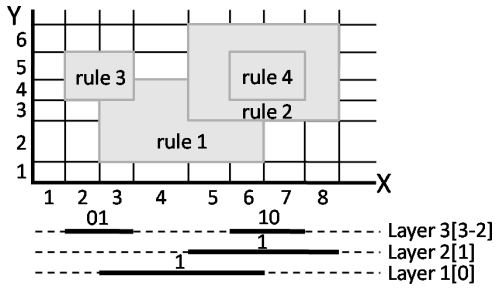


Fig. 1 An example of P^{2}C one-dimension encoding.

the concatenation of multiple one-dimensional coding vectors, resulting in precisely one TCAM entry per rule. The scalability of TCAM both in power consumption and cost could be an issue for applications with large rule tables. This paper presents a multi-dimensional encoding focusing on the multi-dimensional space comprising the source IP address, destination IP address, source port, destination port and protocol type. In this encoding style, rules are divided into several multi-dimensional collision-free rule sets, which are adopted to create the new coding vectors, called Layer Coding Vectors (LCVs), to replace the bit vectors in bitmap intersection algorithm for every dimension. The memory storage of the multi-dimensional encoding is $\Theta(L \cdot N \cdot \log N)$ for each dimension, where L is the number of collision-free rule sets, and N is the number of rules. Under the worst circumstances, the memory storage is still as expensive as bitmap-intersection. However, this study found that the average number of collision-free rule sets in a 2D rule table with 10k rules is only around 14. The multi-dimensional encoding algorithm in average needs much less memory than the bitmap intersection algorithm does. Additionally, the computation required for multi-dimensional encoding is as straightforward as the bitmap intersection algorithm, and does not require as a large TCAM for a large rule table as the P^{2}C approach to achieve high searching speed. Consequently, the proposed new algorithm not only outperforms the bitmap intersection scheme both on memory storage and classification speed, but can also be implemented in a processor without large TCAM. Simulation results of implementation on an Intel IXP 2400 network processor [3] demonstrate that the proposed algorithm not only resolves the issue of large memory requirement in bitmap intersection, but also has a search speed several times that of bitmap intersection.

The remainder of this paper is organized as follows. Section 2 surveys pertinent literature. Section 3 then describes the proposed multi-dimensional encoding algorithm. Next, Sect. 4 summarizes the performance results. Conclusions are finally drawn in Sect. 5.

2. Related Works

The straightforward packet classification algorithm involves a linear search of every rule. The data structure is simple and easily updated to respond to rule changes. Although efficient in terms of memory, a linear search requires a large searching time for a large number of rules. Srinivansan et al. presented a trie-based data structure, called 'Grid of Tries' [4]. A trie is a binary branching tree, with each branch labeled 0 or 1. Grid of trie is a good solution if the rules are limited to only two fields, but can not be applied to more fields to solve the general problem. Thus, Srinivansan et al. [4] consider alternative general solution, called 'Crossproduction,' which generates a table of all possible field value combinations (cross-products), and computes the best-matching rule for each cross-product in advance. Unfortunately, the size of the cross-product table grows significantly with the number of rules and fields. To decrease the memory consumption on the extra rules that are required to represent the cross-products, Dharmapurikar et al. [18] recently presented an architecture solution to apply the "Cross producting Algorithm" to multiple subsets of rules. They introduced an overlay-free grouping to divide the rules into multiple subsets, in a similar manner to the proposed multi-dimensional collision-free rule sets. A cross-product table without additional rules can then be built for every subset. However, to avoid performing multiple lookups in multiple tables, the Bloom filter is introduced to sustain a high throughput as the original cross-producting algorithm.

The 'tuple-space search' [5] partitions the rules into different tuple categories according to the number of specified bits in each dimension and, then, adopts hashing among rules within the same tuple. Tuple-space search has fast average searching and updating time, but is limited in the use of hashing, resulting in lookups or updates of non-deterministic duration.

The Recursive Flow Classification (RFC) [6] presented by Gupta et al. is one of the earliest heuristic approaches. This approach tries to map an S-bit packet header into a T-bit identifier, where $T = \log N$ (N denotes the number of rules) and $T \ll S$. This approach performs the crossproduct in stages, and groups intermediate results into equivalence classes to decrease the storage requirement. RFC works fast, but requires substantial memory, and does not support efficient updating. Gupta et al. also presented another heuristic algorithm, called 'Hierarchical Intelligent Cuttings' (HiCuts) [7], which tries to divide the search space in each dimension, and then builds a decision-tree data structure by carefully preprocessing the rule table. Every leaf node stores a small number of rules, which can be searched linearly to produce the required matching. HiCuts exploits the characteristics of a real rule table, but these characteristics vary. The method for locating an appropriate decision tree prevents good scaling to a larger rule table. The 'Extended Grid of Tries' (EGT) [8] and 'Hyper-Cuts' [9] are general packet classification algorithms that attain high performance without extreme storage space. EGT adopts a slightly modified two-dimensional Grid-of-Tries to classify the source and destination address, followed by a linear search of the rules matching the two fields (Source address and destination address) at that point. HyperCuts, like HiCuts, adopts multidimensional cuts. Unlike HiCuts, in which each node

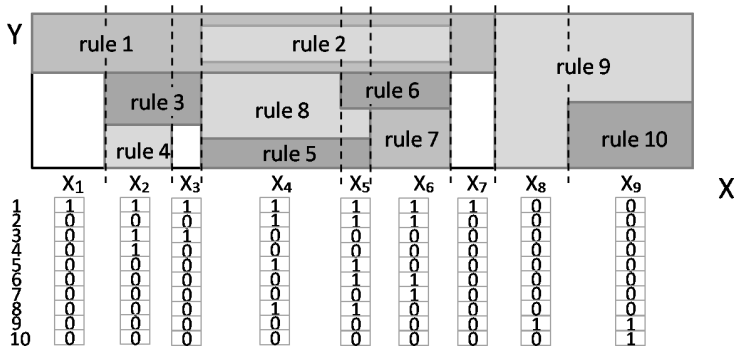


Fig. 2 An example of the bitmap intersection algorithm for a 10 rules set including the corresponding bit vectors in X.

in the decision tree is a hyperplane, each node in the HyperCuts decision tree is a k-dimensional ($k > 1$) hypercube.

Another interesting solution devised by Lakshman et al. [2] is the bitmap intersection algorithm, which adopts the concept of divide-and-conquer, by partitioning the packet classification problem into k sub-problems and then combining the results. This algorithm projects every rule onto each dimension using the geometrical space decomposition. For N rules, a maximum of $2N + 1$ non-overlapping vector intervals are created on each dimension. Every vector interval is associated with an N-bit bit vector. Bit j in the bit vector is set if the projection of the rule range corresponding to rule j overlaps with the vector interval. Figure 2 shows an example of two-dimensional space and the corresponding bitmap for a 2-dimensional rule table. The ten rules are represented by 2-dimensional rectangles. The bitmap intersection first projects the edges of the rectangles to the X-axis, and the ten rectangles then create a total of nine vector intervals. A bit vector is then associated with each vector interval. For instance, the bit vector in vector interval $X2$ is "1011000000," because the first, third, and fourth rules overlap at $X2$. A packet's vector interval for each dimension is found when it arrives. The highest priority entry in a bit vector can be determined from the conjunction of the corresponding bit vectors in each dimension. Because the rules in the rule table are assumed to be sorted in terms of decreasing priority, the first set bit obtained in the resultant bit vector has the highest-priority entry. The rule corresponding to the first set bit is the best matching rule applied to the arriving packet. This scheme employs a bit-level parallelism to match multiple fields concurrently. It can be implemented in hardware to increase the classification speed. However, this algorithm is hard to apply to large rule tables, because the memory storage quadruples with each doubling of the number of rules. The same study describes a variation that decreases the space requirement, while increasing the execution time.

The Aggregated Bit Vector (ABV) algorithm [10], designed by Baboescu et al. represents an improvement on the bit map intersection scheme. Baboescu et al. have made two observations, that sparse set bits are found in the bit vectors, and a packet matches few rules in the rule table. Two key concepts are extended by building on these two observations, namely aggregation of bit vectors and rule rearrangement. Aggregation tries to decrease the memory access time by adding small bit vectors called Aggregate Bit Vector (ABV), which partially capture information from the whole bit vectors. The length of an ABV is given by $\lceil N/A \rceil$, where A represents the aggregate size. Bit k is set in ABV if group k, comprising the bits from position $((k-1)\times A+1)$ bit to $(k \times A)$ in the original bit vector, contains at least one bit set, and is cleared otherwise. Although decreasing the search time for bitmap intersection, aggregation generates another adverse effect, namely false matching, in which the result of conjunction of all ABV returns a set bit, but no actual match exists in the group of rules identified by the aggregate. False matching may lengthen memory access time. Rule rearrangement can eliminate the probability of false matching. Although ABV outperforms bitmap intersection for memory access time by an order of magnitude, uses even more space than bit map intersection. Hence, although ABV has a short memory access time similar to that of multi-dimension encoding algorithm, it requires a larger memory storage than the multi-dimension encoding algorithm does.

Hsu et al. presented the Bit Compression Algorithm (BCA) [11], which is also improves upon the bitmap intersection algorithm. BCA is based on the multiple dimensional range lookup approach. The bit vectors of the bitmap intersection can be compressed, because they include many '0' bits. BCA compresses the bit vectors by preserving only useful information while eliminating the redundant '0' bits of the bit vectors. Additionally, the wildcard rules also enable further improvement in performance. The bit compression algorithm decreases the storage complexity in the average-case from $O(dN^2)$ of bitmap intersection to $\Theta(dN \cdot \log N)$, where d is the number of dimensions, and N is the number of rules. However, BCA requires additional computation time for decompression. The detail survey of packet classification schemes can be found in [13]–[15], [20].

In summary, for the general classification problem with the large number of rules, we find that existing solutions do not balance well between search performance and memory

space requirement. Our paper uses bitmap-intersection as a foundation since it already is proved scalable in search performance. Our scheme adds a new idea by replacing the large size of bit vectors with new coding vectors. The multi-dimensional encoding algorithm was implemented along with bitmap intersection, ABV and bit-compression algorithms on an Intel IXP2400 network processor. This study demonstrates that the multi-dimensional encoding algorithm requires much less memory than bitmap intersection. The throughput performance of the proposed scheme is similar to that of the BCA for the rule tables without wildcard rules and much better than BCA for those with wildcard rules. The multi-dimensional encoding algorithm also performs better than ABV under the rule tables with a large proportion of wildcard rules.

3. Multi-Dimensional Encoding Scheme

As mentioned earlier, bitmap intersection is a hardware-oriented algorithm with rapid classification speed, but suffers from the significant drawback that the storage requirements grow rapidly with the number of rules. The space complexity of bitmap is $O(dN^2)$, where d is the number of dimensions, and N is the number of rules. Although the ABV algorithm has a higher search speed, it requires even more memory space than bitmap intersection. Memory storage is an important performance metric in a hardware solution for packet classification. Reducing the required storage reduces costs correspondingly. The question thus arises whether any method is available to solve the extreme memory storage of a large table while still retaining a straightforward Boolean conjunction in searching performance.

This section introduces a multi-dimensional encoding algorithm requiring less memory storage than the bitmap intersection algorithm. This algorithm divides rules into several multi-dimensional collision-free rule sets, which are utilized to construct the new bit vectors, called layer coding vectors, replacing the bit vectors of the bitmap intersection algorithm. The following sub-section defines the multi-dimensional collision-free rule sets. The layer coding vectors in the proposed algorithm are then introduced. The multi-dimensional encoding algorithm resembles the process of constructing bit vectors in the bitmap intersection algorithm, and is a process of converting multi-dimensional collision-free rule sets into corresponding layer coding vectors. Finally, the proposed multi-dimensional encoding packet classification algorithm is described.

3.1 Multi-Dimensional Collision-Free Rule Set

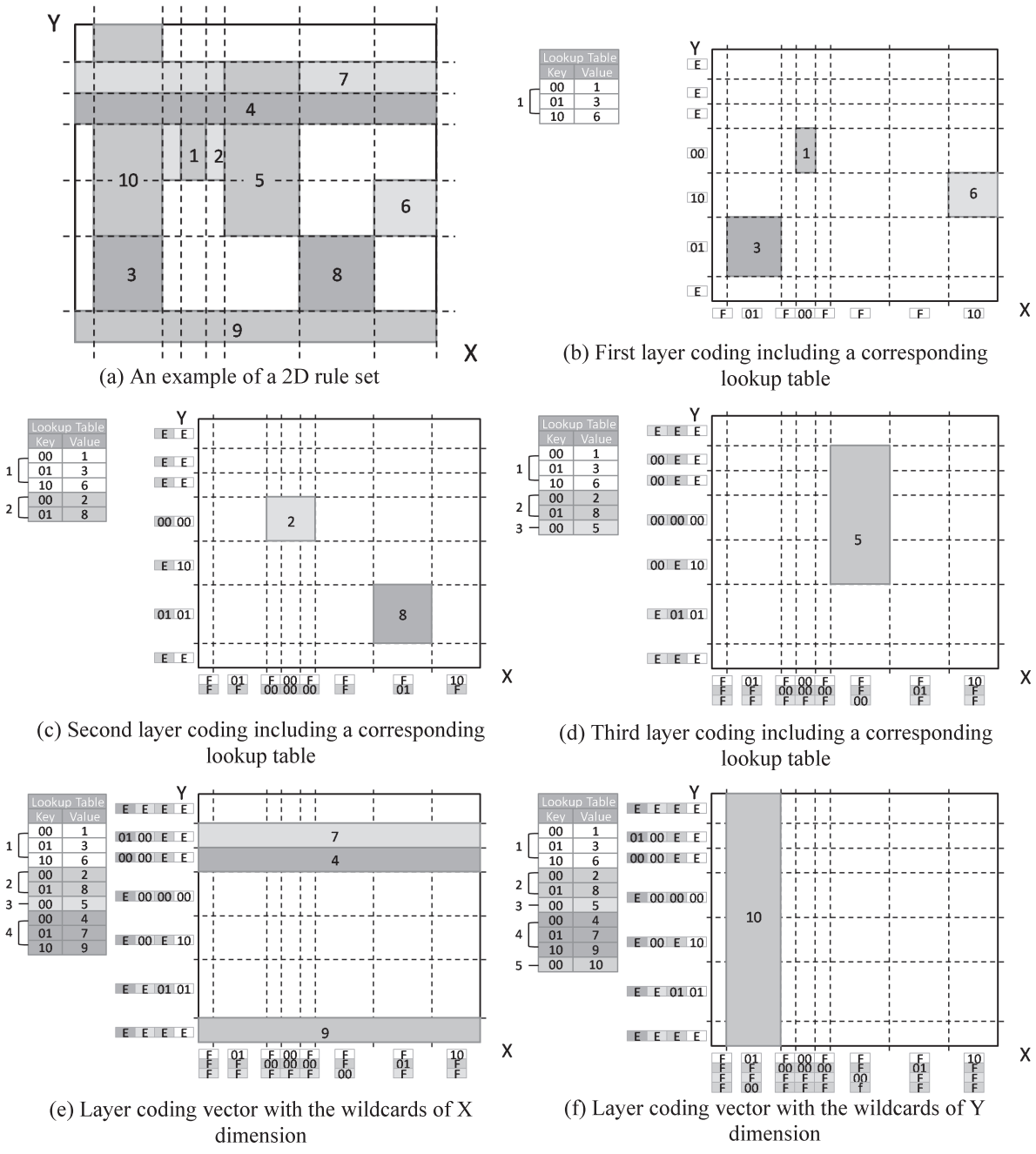
A multi-dimensional collision-free rule set is defined as follows. For a rule R with d fields, let $PR_1, PR_2, \ldots, PR_d$ be primitive ranges of rule R in the dimensions $1, 2, \ldots, d$, respectively. Let $PR_m = [M_p, M_q]$ be the ranges covered in dimension m, where $m \in \{1, 2 \ldots d\}$, and $p \leq q$. Let S be the subset of a rule set, $S = \{R_1, R_2, R_3, R_4, \ldots, R_n\}$. For any R_i, $R_j \in S$, $i \neq j$, the primitive range PR_i is $[M_{ia}, M_{ib}]$, and PR_j is $[M_{jc}, M_{jd}]$ in dimension m. If $[M_{ia}, M_{ib}]$ and $[M_{jc}, M_{jd}]$ do not overlap for any $m \in \{1, 2 \ldots d\}$, then S is called a multi-dimensional collision-free rule set. Significantly, the multi-dimensional collision-free rule sets do not include any wildcard of any dimension. For example, as illustrated in Fig. 3, rules 1, 3, and 6 constitute a 2D collision-free rule set. The set of rules 2 and 8 is also a 2D collision-free rule set.

3.2 Layer Coding Vector (LCV)

The definition of a multi-dimensional collision-free rule set is formally presented above. The Layer Coding Vector (LCV) is now introduced. First of all, the layers specified according to the multi-dimensional collision-free rule sets are defined; one multi-dimensional collision-free rule set forms one layer. Because every layer containing the rules is disjoined in the multi-dimensional search space, every vector interval includes at most one rule in the rule set for any dimension. Hence, every layer forms an element for all LCVs.

In the proposed classification scheme, the classification complexity is proportional to the number of layers. Consequently, minimizing the number of layers is essential to improving performance. However, the number of coding layers is equal to the number of multi-dimensional collision-free sets. The problem of decomposing a multiple dimensional searching space into minimum multi-dimensional collision-free sets can be regarded as a graph-coloring problem [18], which is a NP-hard problem. To decrease the complexity of encoding, this study presents a greedy approach to divide all rules into independent multi-dimensional collision-free rule sets.

Consider the 2D rule table in Fig. 3(a) as an example. First, the wildcards (rules 4, 7, 9, and 10) of the X and Y dimensions are stored temporarily in memory, and then eliminated from the 2D geometrical search space for later processing. The greedy approach is then executed to build the 2D collision-free rule sets. Rule 1 is applied to construct the first layer. Because Rule 2 collides with Rule 1, it cannot be added in the first layer. However, Rules 3 and 1 are collision-free, so Rule 3 is added to the first layer and eliminated from the rule set. Based on this greedy method, Rules 1, 3, and 6 are selected as a collision-free rule set to form the first layer. While forming the first layer, a lookup table is constructed concurrently by adding an entry for each rule in the first layer. Each entry includes a unique key and the rule number. Non-overlapping vector intervals of the two axes are obtained by projecting the rectangle boundaries of rules, including wildcards, onto the corresponding axes as in the bitmap intersection algorithm. Every vector interval maps to a LCV. Every entry of a LCV indicates whether a rule lying in the corresponding vector interval. If so, then the entry is injected with the corresponding unique key of this rule. If not, then a null value is injected. Figure 3(b) illustrates the process of constructing the first layer. Symbols F and E are the null numbers in X and Y dimension respec-



(a) An example of a 2D rule set

(b) First layer coding including a corresponding lookup table

(c) Second layer coding including a corresponding lookup table

(d) Third layer coding including a corresponding lookup table

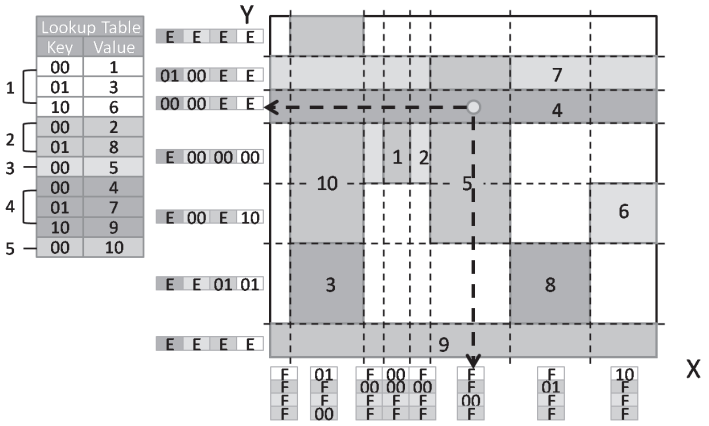
(e) Layer coding vector with the wildcards of X dimension

(f) Layer coding vector with the wildcards of Y dimension

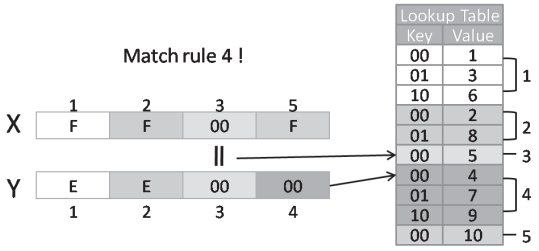
Fig. 3 Illustrate the procedure of multi-dimensional encoding style.

tively. They indicate that no rule exist in the bit vector at this layer. Other layers can be constructed by the same approach. Figures 3(b), 3(c) and 3(d) shown the construction process for each layer. Finally, the wildcards, which are removed temporarily at the beginning of layer construction, of each of the two dimensions are addressed. To minimize memory storage, the wildcards of the *X* dimension are only stored in the LCVs of the *Y* dimension, as shown in Fig. 3(e), since each *X*-dimensional wildcard already covers all intervals in the *X* dimension. Layers have to be built, because wildcards in the *X* dimension may overlap. An identical method can also be applied to the wildcards of the *Y* dimension, as shown in Fig. 3(f).

For the sake of simplicity, the rules ranked in priority according rule number. That is, the rule has the highest priority if its rule number is 1. For a rule table, let IndexTB be a two-dimensional array as a lookup table of LCV, then IndexTB[*i*][*j*] records the rule number, which represents rule *j* in layer *i*. For example, in Fig. 3(e), IndexTB[2][1] indicates the first rule in layer 2, which is rule 2, and IndexTB[1][3] indicates the third rule in layer 1, which is rule 6.



(a) An example of a 2D rules set with corresponding LCVs and lookup table



(b) Comparing the corresponding X and Y LCVs to extract the best-matching rule through lookup table for packet

Fig. 4 Illustrates the procedure of multi-dimension encoding classification algorithm for an incoming packet *P*.

3.3 Multi-Dimensional Encoding Packet Classification Algorithm

The collision-free sets and layer coding vector are defined above. The following paragraphs describe the proposed multi-dimensional encoding classification algorithm.

When a packet *P* arrives, a binary search is run for each dimension on the multi-dimensional search space to obtain the matching vector interval for each dimension. LCVs of the corresponding vector intervals in each dimension are obtained after the binary search is complete. The LCVs are then compard starting from the first element and continuing to the beginning of the wildcard element. We keep track of the matched rule with the highest priority in variable *H*. Moreover, the priority of rule for *H* obtained previously is compared with priorities of wildcards in the corresponding LCVs, and the rule with highest priority is recorded in *H*. Finally, the best matching rule *H* for arriving packet *P* is reported.

Figure 4 illustrates the procedure of the multi-dimensional encoding packet classification algorithm. Figure 4(a) shows an example of a two-dimensional rule set, including its corresponding Layer Coding Vectors. Figure 4(a) shows an incoming packet *P* laid in the two-dimensional search space. To identify the best-matching rule for *P*, the corresponding LCV-*x* "F F 00 F" of the *X*-dimension and LCV-*y* "E E 00 00" of the *Y*-dimension are compared. The value of the first element in LCV-*x* is F(e.g. 1111), which demonstrates that there no rule exists in the first layer of this vector interval in the *X*-dimension, and the value of the first element of LCV-*y* is E (e.g. 1110), which also demonstrates that there no rule exists in the first layer of this vector interval in the *Y*-dimension. The values of the second element in the two LCVs are "F" and "E" respectively, indicating that the second layer of this vector interval contains no rule in the *X* and *Y* dimensions. Subsequently, the values of the third element in the two LCVs are the identical values "00," indicating a matched rule. A search is then performed in the lookup table with "00" and layer number (layer 3) as the key to locate the matched rule number (rule 5). Comparison with *H* indicates that Rule 5 is currently the best-matched rule of packet *P*, so rule 5 is stored in variable *H*. Furthermore, for the wildcard part, that LCV-*x* has the value "F," and LCV-*y* has the value "00." As in the proposed algorithm, rule 4, the wildcards of *X*-dimension are obtained by performing a search in the lookup table. Rule 4 has a higher priority than Rule 5, so Rule 4 is stored in *H*. Finally, Rule 4 is the best-matching rule for the packet *P*.

Figure 5 shows the data structures of the layer coding vectors for a 2D rule table. Every vector consists of two parts of layer coding. The first part is the layer coding for the non-wildcard rules. The second part is the layer coding for the "do not care" rules in either the *X*- or *Y*-dimension. Although so far only the examples of a multi-dimensional encoding algorithm working on a two-

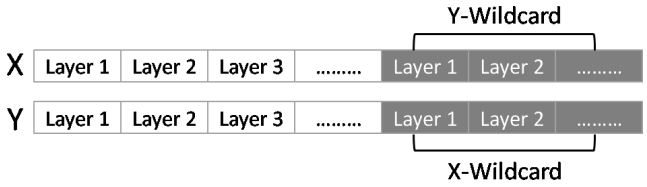


Fig. 5 The data structure of LCVs in two-dimensional encoding style.

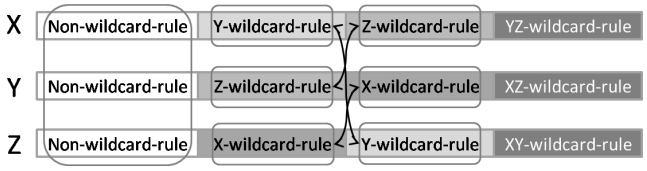


Fig. 6 The data structure of LCVs in three-dimensional encoding style.

dimensional space are shown, the algorithm can extend to a multi-dimensional search space. Considering a three-dimensional case as an example, Fig. 6 shows the data structure of the layer coding vectors. After 3D collision-free sets are found, three layer coding vectors corresponding to the *X*-, *Y*-, and *Z*-dimensions are built. Furthermore, every LCV includes three parts of layer coding for the non-wildcard rules, for rules with "don't care" only in one of the dimensions, and for rules with "don't care" in both dimensions. When the multi-dimensional encoding algorithm performs packet classification, it compares the layers with the same property to discover the matched rules as in the two-dimensional case. As shown in Fig. 6, the first layer for non-wildcard rules is compared first among X, Y, and Z LCVs. The second part of layer coding in each dimension's LCV is then compared only with the second part of layer coding of the other dimensions' LCVs, which has the rules with "don't care" in the same dimension. The arrowed edges in Fig. 6 demonstrate that both layer coding segments of LCVs have the wildcard rules in the same dimension. Finally, the third part of layer coding is compared among all dimensions. The highest-priority matching rule from those comparisons is reported as the best matching rule. Similarly, a greedy algorithm can be applied to obtain the multi-dimensional collision-free rule sets in the multi-dimensional search space; the proposed algorithm creates the different LCVs for different dimensions. When a packet arrives, the binary search is applied to obtain the corresponding LCVs for each dimension. The same kinds of layers of LCVs in all dimensions are then compared to obtain the best matching rule. The pseudocode of the two-dimension classification procedure is shown below:

```
1.  Binary search to find the position of packet on the bit-map
2.  Read the associated d LBVs from memory, if d is the number of the dimensions
    The layer number i which we currently check is assigned to 1.
    i←1
    Maintain a register H to record the currently best-matched rule.
3.  If the values of the ith entry in LBV of X-dimension is equal to M and not equal to 1111(F)
    and the values of the ith entry in LBV of Y-dimension is equal to N and not equal to 1110(E)
        and M=N then
        IndexTB[ i ][ j ] is the matched rule of the layer i.
        If ( IndexTB[ i ][ M ] <H )
            H←IndexTB[ i ][M]
4.  i++
5.  Repeate step 3 to 4 until i reaching the wildcards of the layer
6.  If the values of the LBV in the X-dimension is equal to M and M not equal to 1111(F),
    IndexTB[ i ][ M ] is the matched rule of the layer i.
    If ( IndexTB[ i ][ M ] <H )
        H←IndexTB[ i ][ M ]
7.  If the values of the LBV in the Y-dimension is equal to N and N not equal to 1110(E),
    IndexTB[ i ][ N ] is the matched rule of the layer i.
    If ( IndexTB[ i ][ N ] <H )
        H←IndexTB[ i ][ N ]
8.  H is the best-matched rule.
```

4. Number of Layers Analysis

The space requirement of the multi-dimension encoding algorithm consists of two components, the LCVs and the lookup table. This study ignores the space complexity of the lookup table, since it is far smaller than the LCVs. Because the number of layer coding vectors for each dimension is proportional to the number of rules, the depth of each layer's coding vector is related to the number of layers, and the size of encoding bits is much less than $\log N$, so the average memory space complexity can be obtained as Θ (d. L. N. $\log N$), where d is the number of dimensions, L is the number of collision-free rule sets, and N is the number of rules.

We know that the performance of our algorithm can deteriorate significantly depending on the rule patterns. For example, if all the rules are overlapping with each other (i.e. every rules overlaps with every other rule). Then the number of layers needed is equal to the number of rules, and our algorithm results in a linear search. To test the number of layers on the rule tables with hundred thousands of rules, we first synthesize the two-dimensional rule table artificially with 1 K, 2 K, 3 K . . . 10 K rules. In this approach, the rule table is constructed from the prefix length distribution probability based on five publicly available routing tables [10] and the probability, β, [17], which denotes the probability that prefix P_A is a prefix of prefix P_B, where P_A and P_B are randomly selected from the rule table. β is an important parameter for controlling rule overlapping probability. Overlapping probability rises with increasing β. In [17], the authors derived the value of β, and acquired the results of about 10^{-5} for several real-life routing tables obtained from the Mae-West routing database. In [19], the authors indicated that the number of overlaps in overlapping IP address pairs is significantly smaller than the theoretical upper limit.

Since the proposed algorithm does not alter the number of bit vectors in bit intersection, the saved storage space is affected by the number of layers L. Because L equals the number of multi-dimensional collision-free sets, the number of collision-free sets in the synthetic source-destination IP address tables under different values of β (10^{-3}, 10^{-4} and 10^{-5}) was examined. Table 1 lists the statistical results of the synthetic approach. Table 1 is obtained by apply-

Table 1 β versus the average number of layers.

# of rules	β=10^{-3}	β=10^{-4}	β=10^{-5}
1k	10.94	4.65	3.17
2k	18.03	5.98	3.54
3k	24.96	6.96	3.92
4k	32.79	7.92	4.14
5k	3.18	8.95	4.3
6k	45.01	9.94	4.63
7k	53.47	10.92	4.85
8k	58.37	11.7	5.05
9k	64.85	12.33	5.1
10k	71.21	13.22	5.46

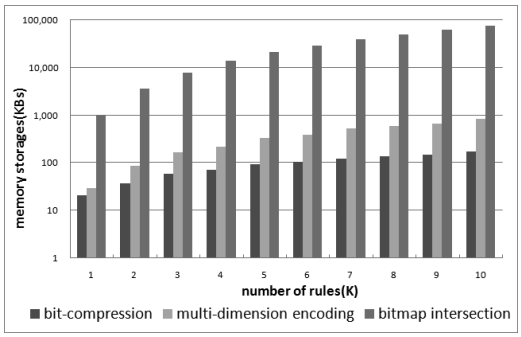


Fig. 7 Memory requirements vs. number of rules.

ing the greedy approach described in Sect. 3. A to count the collision-free rule sets for all synthetic rules. The average values of collision-free sets of 1000 statistics were computed. The number of collision- free sets rose with β for rule tables created with this synthetic approach, as expected. Table 1 demonstrates that the ratio of average number of collision-free sets to rule numbers remains very low (below 0.01) even with large β. The statistics confirm that the proposed algorithm saves significant storage comparing with the bitmap intersection.

5. Performance Results

In our memory requirement and throughput experiment, the 2D rule tables were generated by the synthetic approach with $\beta = 10^{-4}$. The multi-dimensional encoding, bitmap intersection, ABV, and bit compression algorithms were implemented using Micro C. Experiments were undertaken on an Intel IXP2400 Developer Workbench. Intel IXP2400 is a network processor with eight processors called Microengines (ME), giving it a high degree of parallelism. Every ME has eight hardware-assisted threads running at 600 MHz. The complexity of storage requirement and classification performance was considered in this study. The proposed multi-dimensional encoding algorithm was compared with the bitmap intersection, bit-compression and aggregated bit vector algorithms.

This performance study focused on the two-dimensional rule table, and so all algorithms were optimized to handle only IP destination and source addresses. The memory requirements of the bitmap-based algorithms are linear with the number of dimensions. Furthermore, the query times of the bitmap-based algorithms are bounded by the times they take to access the bitmaps in memory, which is again linear with the number of dimensions. Therefore, the performance evaluation results for two-dimensional rules can be extended to multiple-dimension cases. The priority was based on the order of rules (although the ABV implementation uses a different sorting method). The bitmap intersection and bit compression algorithms are both designed to find the first matched rule which is also the highest priority rule, and therefore do not need to read an entire bit vector. The proposed multi-dimensional encoding algorithm adopts a fixed-length design for coding keys, extending each coding key by 16 bits. This design enables each layer to obtain a fixed code size, significantly improving its efficiency in the decoding stage. Figure 7 shows the memory storage requirement for the three algorithms. Although the memory space required by multi-dimensional encoding is larger than that required by bit-compression, it can still be placed in the SRAM of IXP2400. Unlike bitmap intersection, in which 10k rules need almost 70 MB, multi-dimensional encoding only requires 800 KB of space. Although, a variable length encoding method can further decrease storage requirement, it will increase the decoding time.

We implement the multi-dimensional encoding algorithm along with bitmap intersection, ABV and bit-compression algorithms with Microengine C. Experiments are conducted on the Intel IXP 2400 Developer Work-Bench. The throughput is measured by the data arriving rate (i.e. link transmission rate) with packet size equal to minimum packet size (46 Bytes). Figures 8, 9, 10, and 11 show the transmission rates of the four algorithms under different rule tables sizes with 0%, 10%, 30% and 50% wildcards. As demonstrated in Fig. 8, the bit-compression, multi-dimension encoding, and ABV all performed better than bitmap-intersection in the rule tables without wildcards.

To lower the probability of false matching in ABV, a new sorting scheme has been presented to concentrate the set bits in the bit vectors produced by the rules. However, the experiments indicate that reordering the 1st dimension decreased the concentration of the bits in the 2nd dimension. Hence, ABV's performance fell in respect to the rule table with a large percentage of wildcards, because a large percentage of wildcards raises the chance of false matching. However, multi-dimensional encoding increases its performance with more wildcard rules, because if more wildcards are given, then the total number of layers is decreased. As revealed in Figs. 9, 10 and 11, we see that the performance gap widened between multi-dimensional encoding and ABV when the number of wildcard rules rose.

The experimental results shown in Fig. 8 indicate that bit compression algorithm has a similar performance to multi-dimensional encoding when rules do not have wildcards. Both these algorithms outperformed bitmap intersec-

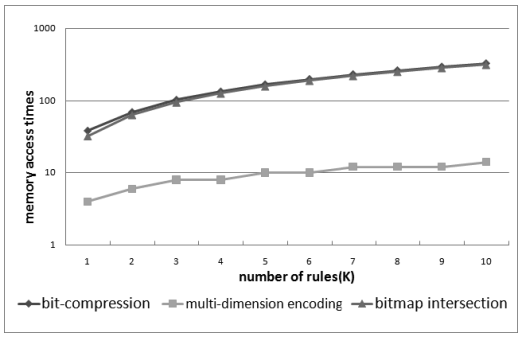


Fig. 8 Transmission rate vs. number of rules with 0% wildcard.

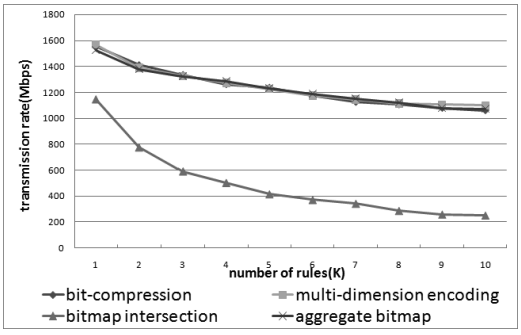


Fig. 9 Transmission rate vs. number of rules with 10% wildcards.

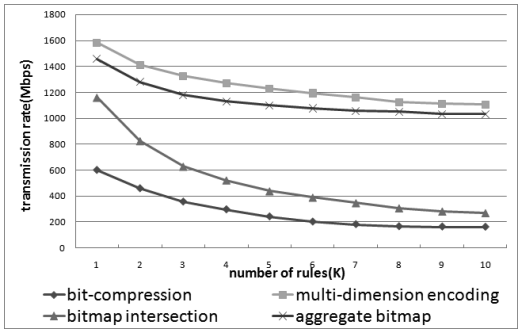


Fig. 10 Transmission rate vs. number of rules with 30% wildcards.

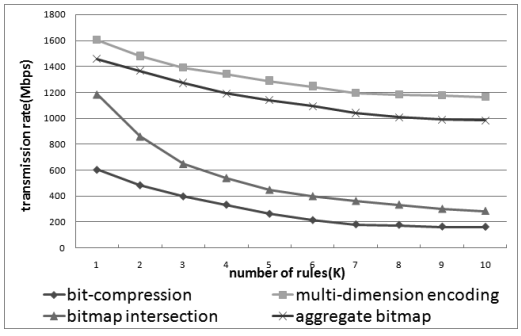


Fig. 11 Transmission rate vs. number of rules with 50% wildcards.

tion algorithms in these experiments. As previously mentioned, DCV is applied in bit compression to reserve the wildcard information if a rule table is considered to have

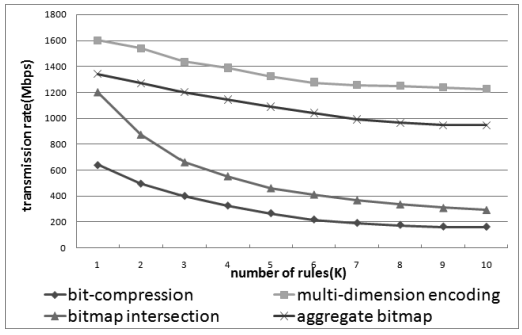


Fig. 12 # of memory access vs. number of rules with 10% wildcards.

wildcards. Therefore, in practice, DCV can be omitted, and bit compression scheme does not have to access DCV if a rule table does not contain wildcards. However, rule tables with 10%, 30%, and 50% wildcards produced different simulation results, as demonstrated in Figs. 9, 10 and 11, respectively. The results indicate, as expected that the bit compression algorithm performed worse than multi-dimensional encoding, ABV, and even bitmap intersection algorithm, due to the additional time needed to access DCVs, which is proportional to the size of the rule table. Figure 12 confirms that the bit compression algorithm has an even larger number of memory access than the original bitmap intersection algorithm when the rule tables contain 10% wildcards. The query time of the proposed multi-dimension encoding algorithm includes the time needed for binary search, the time for LCVs access and the time for index table lookup. Since the number of vector intervals is the same for all algorithms, the time needed for binary search is the same. Therefore, the performance gains of our algorithms resulted mainly from the low memory access requirement of LBVs.

6. Conclusion

Packet classification is a fundamental function of Internet security, virtual private networks, QoS and several other network services. Many studies have considered this problem. This study tries to enhance the original bitmap intersection and P^2C algorithms, which have a memory explosion problem and a large TCAM requirement problem respectively for large rule tables. This study introduces the concept of multi-dimensional encoding to significantly reduce the storage requirement. The proposed algorithm divides rules into several collision-free rule sets, which are utilized to construct the new layer coding vectors, replacing the bit vector in bitmap intersection algorithm. The storage complexity is reduced from $O(d \cdot N^2)$ of bitmap intersection to $\Theta(d \cdot L \cdot N \cdot \log N)$ in average. This study has found that the average number of collision-free rule sets in a 2D rule table with 10k rules is only around 14 with $\beta = 10^{-4}$. Experimental results indicate that the proposed multi-dimensional encoding algorithm requires less than 800 kbytes to store the 2-dimensional rule table with 10 K rules, while bitmap intersection requires

70 Mbytes. Additionally, by exploiting the memory hierarchy of IXP 2400, our multi-dimensional encoding algorithm needs much less memory access time than bitmap intersection and bit-compression. Although the multi-dimensional encoding scheme requires additional processing time for decoding when performing multi-dimensional encoding, it still outperforms bitmap intersection, bit-compression and even ABV in terms of classification speed.

Acknowledgment

This research was supported by the New Generation Broadband Wireless Communication Technologies and Applications Project of Institute for Information Industry and sponsored by MOEA, R.O.C.

References

[1] M.H. Overmars and A.F. van der Stappen, "Range searching and point location among fat objects," J. Algorithms, vol.21, no.3, pp.629–656, Nov. 1996.
[2] T.V. Lakshman and D. Stiliadis, "High-speed policy-based packet forwarding using efficient multi-dimensional range matching," Proc. ACM Sigcomm, pp.191–202, Sept. 1998.
[3] Intel® IXP2400 Network Processor: Flexible, High-Performance Solution for Access and Edge Applications White Paper, 2002.
[4] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and scalable layer four switching," Proc. ACM Sigcomm, pp.203–214, Sept. 1998.
[5] V. Srinivasan, S. Suri, and G. Varghese, "Packet classification using tuple space search," Proc. ACM Sigcomm, pp.135–46, Sept. 1999.
[6] P. Gupta and N. McKeown, "Packet classification on multiple fields," Proc. ACM Sigcomm, pp.147–160, Sept. 1999.
[7] P. Gupta and N. McKeown, "Packet classification using hierarchical intelligent cuttings," IEEE Micro, vol.20, no.1, pp.34–41, Jan./Feb. 2000.
[8] F. Baboescu, S. Singh, and G. Varghese, "Packet classification for core routers: Is there an alternative to CAMs," Proc. IEEE Infocom, vol.1, pp.53–63, March 2003.
[9] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," Proc. ACM Sigcomm, pp.213–224, Aug. 2003.
[10] F. Baboescu and G. Varghese, "Scalable packet classification," Proc. ACM Sigcomm, pp.199–210, Aug. 2001.
[11] C.R. Hsu, C. Chen, and C.Y. Lin, "Fast packet classification using bit compression," Proc. IEEE Globecom, vol.2, pp.739–743, Nov./Dec. 2005.
[12] J. Lunteren and T. Engbersen, "Fast and scalable packet classification," IEEE J. Sel. Areas Commun., vol.21, no.4, pp.560–571, May 2003.
[13] P. Gupta and N. McKeown, "Algorithms for packet classification," IEEE Netw., vol.15, no.2, pp.24–32, March/April 2001.
[14] D. Taylor, "Survey & taxonomy of packet classification techniques," Washington University Technical Report, WUCSE-2004-24, 2004.
[15] G. Varghese, Network algorithmics: An interdisciplinary approach to designing fast networked devices, Morgan Kaufmann Publishers, San Francisco, CA, 2005.
[16] http://www.merit.edu/ipma/routing_table
[17] G. Zhang, H.J. Chao, and J. Joung, "Fast packet classification using field-level trie," Proc. IEEE Globecom, vol.6, pp.3201–3205, Dec. 2003.
[18] S. Dharmapurikar, H. Song, J. Turner, and J. Lockwood, "Fast packet classification using bloom filters," Washington University Technical Report, WUCSE-2006-27, May 2006.
[19] M.E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A.T. Campbell, "Directions in packet classification for network processors," Proc. Second Workshop on Network Processors (NP2), Feb. 2003.
[20] C.J. Huang, C. Chen, C.S. Chou, and S.T. Kao, "Fast packet classification using multi-dimensional encoding," Proc. Workshop on High Performance Switching and Routing, May 2007.

Chi Jia Huang received his B.S. degree in Computer Science from National Chiao Tung University in 2004. He is a master student in Department of Computer Science and Information Engineering of National Taiwan University now. His current research interests include artificial intelligent and computer networks.

Chien Chen received his B.S. degree in Computer Engineering from National Chiao Tung University in 1982 and the M.S. and Ph.D. degrees in Computer Engineering from University of Southern California and Stevens Institute of Technologies in 1990 and 1996. Dr. Chen hold a Chief Architect and Director of Switch Architecture position in Terapower Inc., which is a terabit switching fabric SoC startup in San Jose, before joining National Chiao Tung University as an Assistant Professor in August 2002. Prior to joining Terapower Inc., he is a key member in Coree Network, responsible for a next-generation IP/MPLS switch architecture design. He joined Lucent Technologies, Bell Labs, NJ, in 1996 as a Member of Technical Staff, where he led the research in the area of ATM/IP switch fabric design, traffic management, and traffic engineering requirements. His current research interests include wireless ad-hoc and sensor networks, switch performance modeling, and DWDM optical networks.